**Data Manipulation Language (DML)**

DML stands for Data Manipulation Language, and it refers to a set of SQL commands used to manage data within a database. The commonly used DML commands in SQL are:

SELECT: Retrieves data from one or more tables or views based on specified criteria.

INSERT: Adds new rows of data into a table.

UPDATE: Modifies existing data in a table.

DELETE: Removes rows of data from a table.

These DML commands form the core functionality for interacting with and managing data within a SQL database. They allow you to query, insert, update, and delete data to maintain the integrity and consistency of your database.

The INSERT INTO command in SQL is used to add new rows of data into a table. It allows you to specify the table name, the columns into which you want to insert data, and the values to be inserted.

* Definition: The INSERT INTO command is a Data Manipulation Language (DML) statement in SQL that adds new records to a table.
* Syntax: The basic syntax of the INSERT INTO command is as follows:

INSERT INTO TableName (Column1, Column2, ...)

VALUES (Value1, Value2, ...);

Parameters:

* TableName: The name of the table where you want to insert data.
* (Column1, Column2, ...): The list of columns in the table where you want to insert data.
* VALUES: The keyword indicating the start of the list of values to be inserted.
* (Value1, Value2, ...): The list of values to be inserted into the specified columns.

Examples:

Inserting a single row of data into a table with all columns specified:

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary)

VALUES (1, 'John', 'Doe', 101, 50000);

Inserting multiple rows of data into a table with only specific columns specified:

INSERT INTO Employees (FirstName, LastName)

VALUES ('Alice', 'Smith'), ('Bob', 'Johnson'), ('Charlie', 'Brown');

Inserting data into a table with values derived from a subquery:

INSERT INTO NewTable (Column1, Column2, ...)

SELECT Column1, Column2, ...

FROM ExistingTable

WHERE Condition;

Inserting default values into all columns of a table:

INSERT INTO Employees DEFAULT VALUES;

Copying data from one table into another table:

INSERT INTO TargetTable (Column1, Column2, ...)

SELECT Column1, Column2, ...

FROM SourceTable;

These examples demonstrate the versatility of the INSERT INTO command in SQL, allowing you to insert data into tables in various ways, including single or multiple rows, with specified columns or default values, and even from other tables or subqueries.

The UPDATE command in SQL is used to modify existing records in a table. It allows you to change the values of one or more columns in one or more rows based on specified conditions.

* Definition: The UPDATE command is a Data Manipulation Language (DML) statement in SQL that modifies existing records in a table.
* Syntax: The basic syntax of the UPDATE command is as follows:

UPDATE TableName SET Column1 = Value1, Column2 = Value2, ... WHERE Condition;

Parameters:

* TableName: The name of the table where you want to update records.
* SET: The keyword indicating the columns to be updated and their new values.
* Column1 = Value1, Column2 = Value2, ...: The list of columns and their corresponding new values.
* WHERE: The keyword indicating the start of the condition to filter the rows to be updated.
* Condition: The condition that must be met for the update operation to be applied to the rows.

Examples:

Update a single column in all rows of a table:

UPDATE Employees SET Salary = Salary \* 1.1;

Update multiple columns in a specific row based on a condition:

UPDATE Employees SET FirstName = 'Robert', LastName = 'Smith', Salary = 60000

WHERE EmployeeID = 1;

Update data in a table using values from another table or subquery:

UPDATE Employees SET DepartmentID = (SELECT DepartmentID FROM Departments

WHERE DepartmentName = 'IT') WHERE DepartmentID IS NULL;

Update data based on multiple conditions:

UPDATE Students SET MajorID = 102 WHERE MajorID = 101 AND GPA > 3.5;

Update data in a table using the results of a JOIN operation:

UPDATE Orders SET OrderStatus = 'Shipped' FROM Orders

JOIN Customers ON Orders.CustomerID = Customers.CustomerID

WHERE Customers.Country = 'USA';

These examples demonstrate the flexibility of the UPDATE command in SQL, allowing you to update data in tables based on various conditions, including simple column-value pairs, complex conditions, subqueries, and JOIN operations.

The DELETE command in SQL is used to remove one or more rows from a table based on specified conditions. It permanently removes data from the table.

* Definition: The DELETE command is a Data Manipulation Language (DML) statement in SQL that removes existing records from a table.
* Syntax: The basic syntax of the DELETE command is as follows:

DELETE FROM TableName WHERE Condition;

Parameters:

* TableName: The name of the table from which you want to delete rows.
* FROM: The keyword indicating the start of the table from which to delete rows.
* WHERE: The keyword indicating the start of the condition to filter the rows to be deleted.
* Condition: The condition that must be met for the delete operation to be applied to the rows.

Examples:

Delete all rows from a table:

DELETE FROM Employees;

Delete specific rows based on a condition:

DELETE FROM Employees WHERE DepartmentID = 101;

Delete a single row based on a unique identifier:

DELETE FROM Employees WHERE EmployeeID = 1;

Delete rows from a table using values from another table or subquery:

DELETE FROM Employees

WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'HR');

Delete rows from a table using a join condition:

DELETE Orders FROM Orders JOIN Customers ON Orders.CustomerID = Customers.CustomerID WHERE Customers.Country = 'France';

These examples demonstrate the usage of the DELETE command in SQL, allowing you to remove rows from a table based on various conditions, including simple column-value pairs, complex conditions, subqueries, and JOIN operations. It's important to use caution when using the DELETE command as it permanently removes data from the database.

UPDATE Examples:

Increase Salary by 10% for All Employees:

UPDATE Employees SET Salary = Salary \* 1.1;

Change DepartmentID for Employees with Null Department:

UPDATE Employees SET DepartmentID = 100 WHERE DepartmentID IS NULL;

Update LastName for Employees with Specific FirstName:

UPDATE Employees SET LastName = 'Doe' WHERE FirstName = 'John';

Promote Employees with Tenure Over 5 Years:

UPDATE Employees SET JobTitle = 'Senior ' + JobTitle WHERE Tenure > 5;

Increment Salary for Employees in IT Department:

UPDATE Employees SET Salary = Salary \* 1.05 WHERE DepartmentID = 101;

DELETE Examples:

Remove Employees with Salary Less Than 30000:

DELETE FROM Employees WHERE Salary < 30000;

Delete Orders Placed Before January 1, 2022:

DELETE FROM Orders WHERE OrderDate < '2022-01-01';

Remove Inactive Customers:

DELETE FROM Customers WHERE LastPurchaseDate < '2021-01-01';

Delete All Records from a Table:

DELETE FROM TableName;

Remove Students with GPA Less Than 2.0:

DELETE FROM Students WHERE GPA < 2.0;

**Transaction Control Languages**

**TCL** stands for **Transaction Control Languages**. These commands are used for maintaining consistency of the database and for the management of transactions made by the DML commands.   
  
A **Transaction** is a set of SQL statements that are executed on the data stored in DBMS. Whenever any transaction is made, these transactions are temporarily happen in database; so to make the changes permanent, we use **TCL** commands.

Applications of TCL:

1. Committing Transactions: TCL statements can be used to commit a transaction, which means to permanently save the changes made during the transaction to the database.
2. Rolling Back Transactions: TCL statements can be used to roll back a transaction, which means to undo the changes made during the transaction and restore the database to its previous state.
3. Savepoints: TCL statements can be used to set savepoints within a transaction, allowing for partial rollback if needed.
4. Managing Transactions in Stored Procedures: TCL statements can be used in stored procedures to manage transactions within the scope of the procedure.

Overall, TCL is an essential part of SQL and is used extensively in database management systems to control transactions and ensure data consistency. By using TCL statements, database administrators and developers can manage transactions effectively and maintain the integrity of their databases.

The TCL commands are:

1. COMMIT
2. ROLLBACK
3. SAVEPOINT

**1. COMMIT:**   
This command saves all changes made during the current transaction permanently to the database. Whenever we perform any of the DML command like -INSERT, DELETE or UPDATE, these can be rollback if the data is not stored permanently. So, in order to be at the safer side COMMIT command is used.   
  
**Syntax:**

commit [transaction];

**2. ROLLBACK :**   
This command is used to get the data or restore the data to the last savepoint or last committed state. If due to some reasons the data inserted, deleted or updated is not correct, you can rollback the data to a particular savepoint or if savepoint is not done, then to the last committed state.   
  
**Syntax:**

Rollback [transaction];

**3. SAVEPOINT :**   
This command is used to save the data at a particular point temporarily, so that whenever needed can be rollback to that particular point.

**Syntax:**

Savepoint savepointname;

Example:

Employee table:

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Salary DECIMAL(10, 2)

);

Insert records:

INSERT INTO Employee (EmployeeID, FirstName, LastName, Salary)

VALUES (1, 'John', 'Doe', 50000),

(2, 'Jane', 'Smith', 60000),

(3, 'Alex', 'Johnson', 55000);

And then:

* select \* from Employee;
* truncate table Employee;
* BEGIN TRANSACTION;
* UPDATE Employee SET Salary = Salary + 1;
* Rollback transaction;
* COMMIT;
* COMMIT TRANSACTION;
* IF @@ERROR <> 0

ROLLBACK TRANSACTION;

ELSE

COMMIT TRANSACTION;

And, for savepoint

* BEGIN TRANSACTION;
* UPDATE Employee SET Salary = Salary + 1;
* SAVE TRANSACTION Step1;
* UPDATE Employee SET Salary = Salary + 1;
* SAVE TRANSACTION Step2;
* UPDATE Employee SET Salary = Salary + 1;
* SAVE TRANSACTION Step3;
* UPDATE Employee SET Salary = Salary + 1;
* SAVE TRANSACTION Step4;
* UPDATE Employee SET Salary = Salary + 1;
* SAVE TRANSACTION Step5;
* UPDATE Employee SET Salary = Salary + 1;
* SAVE TRANSACTION Step6;
* rollback transaction Step2

Note: when you rollback to a savepoint, the savepoints declared after that are not existing and accessible anymore; i.e. if for above example you rollback to step4, then step5 and step6 are not existing anymore.